

# A GPU-based High-Performance Library with Application to Nonlinear Water Waves

**Glimberg, Stefan Lemvig; Engsig-Karup, Allan Peter**

*Publication date:*
2012

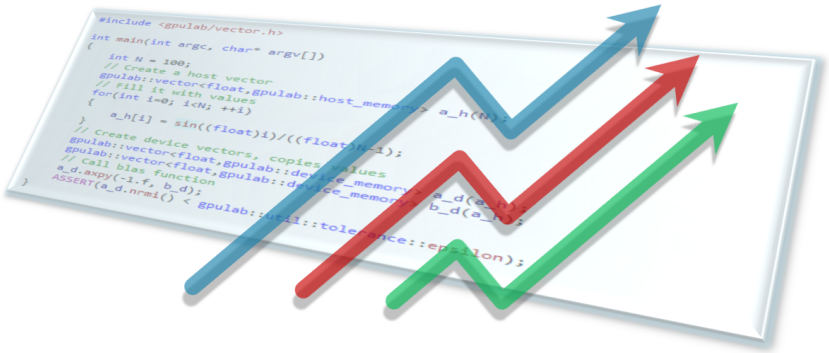
*Document Version*
Publisher's PDF, also known as Version of record

# A GPU-based High-Performance Library with Application to Nonlinear Water Waves

PhD student: Stefan L. Glimberg
Supervisor: Assoc. Prof. Allan P. Engsig-Karup

Technical University of Denmark
Department of Informatics and Mathematical Modelling
Section of Scientific Computing

DANSIS Research Seminar
May 23$^{rd}$, 2012

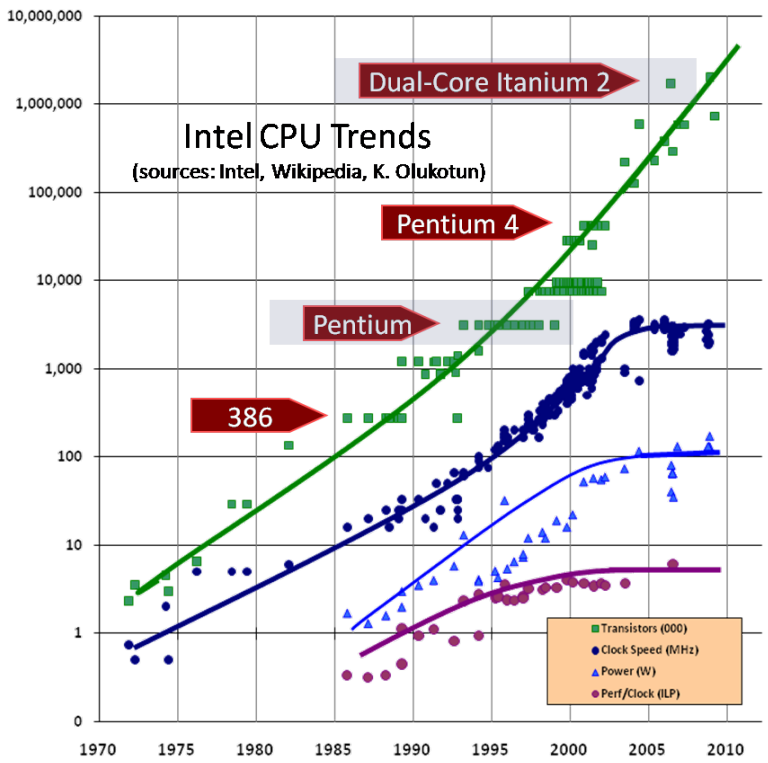


GPUlab DTU Informatics

# High-Performance Heterogeneous Computing

## Moore's Law [1965]

Chip performance doubles every two years ...



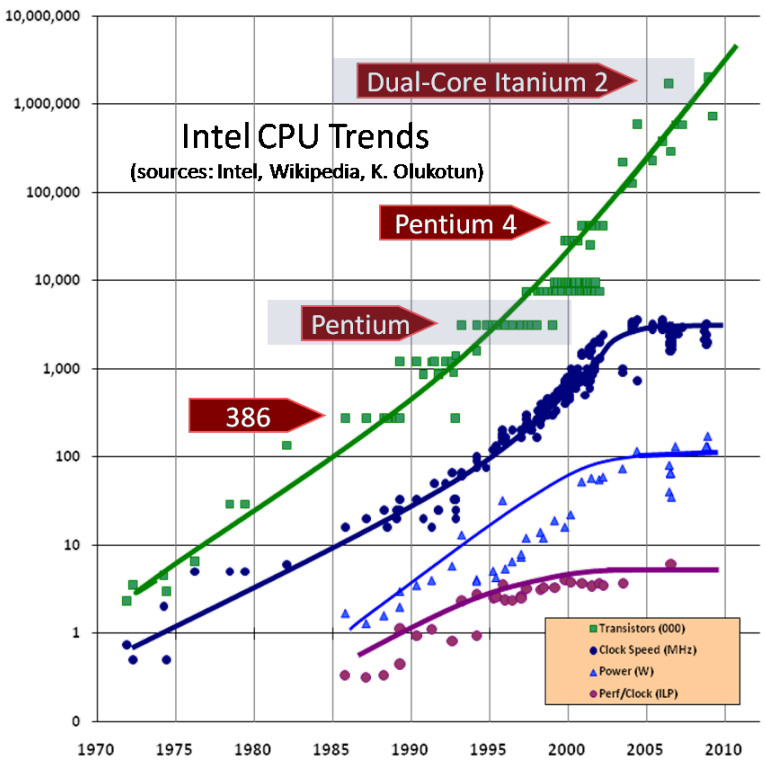

Physical limitations caused clock speed to stall around 10 years ago.

The free lunch is over!

# High-Performance Heterogeneous Computing

## Moore's Law [1965]

Chip performance doubles every two years ...




Physical limitations caused clock speed to stall around 10 years ago.

The free lunch is over!

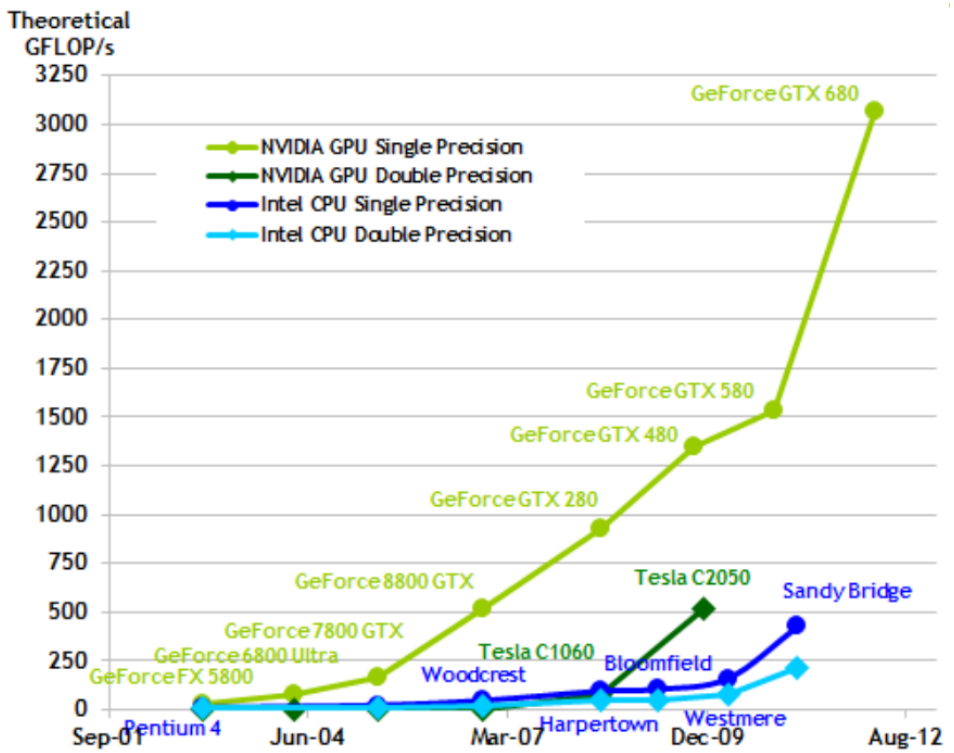
Solutions: Hyper-threading, multi-core (CPUs), many-core co-processors (GPUs).

GPUlab DTU Informatics

# Motivation for CPU/GPU Heterogeneous Computing

Some good reasons to consider Graphical Processing Units for high-performance computing

- Massively parallel architecture (1,536 cores in Kepler GK104)
- Teraflop performance
- Moderate prices \$100 – \$2,000, buy a personal super computer
- High-level programming models (CUDA, OpenCL)
- Number 2, 4, and 5 fastest supercomputers are based on GPUs



# A GPU-based Framework for PDE Solvers

We have build a highly generic heterogenous CPU-GPU framework for fast PDE solver prototyping.

## Framework Objectives

- Remove GPU-specific code for the non-GPU-expert programmer
- While maintaining the possibility to customize code at kernel level

```
1  gpulab::vector<float,host_memory>    x_h(100,3.f); // Create host vector x, size 100, value 3
2  gpulab::vector<float,device_memory>  x_d(x_h);     // Create device vector x, transfer host data
3  gpulab::vector<float,device_memory>  y_d(x_d);     // Create device vector y, copy device data
4  y_d.axpy(4.f,x_d);                                 // Do y = a*x+y on the device
5  float n = y_d.nrm2();                              // Calculate the 2-norm on the device
```

Ideas are based on the C++ standard library, and Thrust/CUSP that exist for CUDA.

GPUlab DTU Informatics

# Library Components

The library contains several useful components for assembling PDE solvers.

## Components

- Vectors (CPU / GPU)
- Regular grids, 1D, 2D, 3D.
- Compact flexible order finite difference operators (matrix free)
- Iterative solvers for solving linear system of equations, CG, GMRES, Defect Correction, Multigrid.
- I/O vector operations

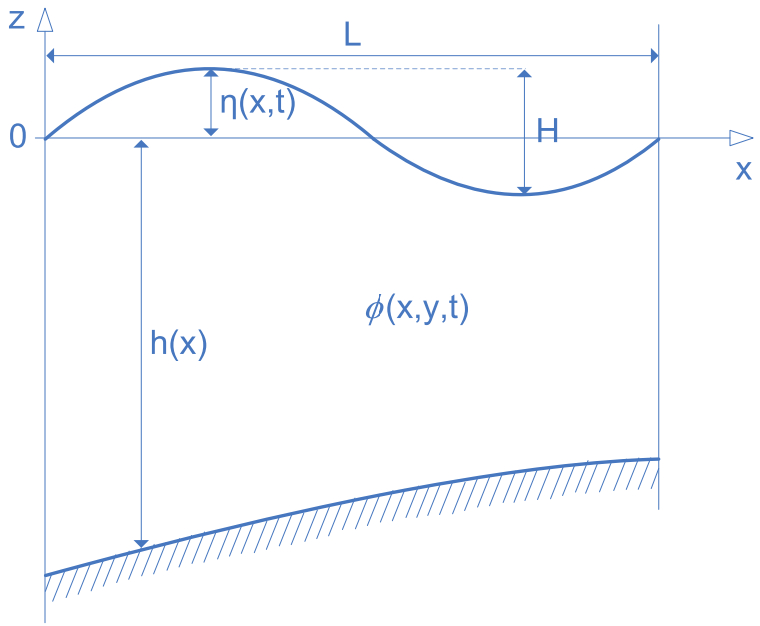
Template-based implementations enable end-users to define their own user-specific implementations.

GPUlab DTU Informatics

# Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of inviscid and irrotational flow.

## 2D Potential Flow Equations



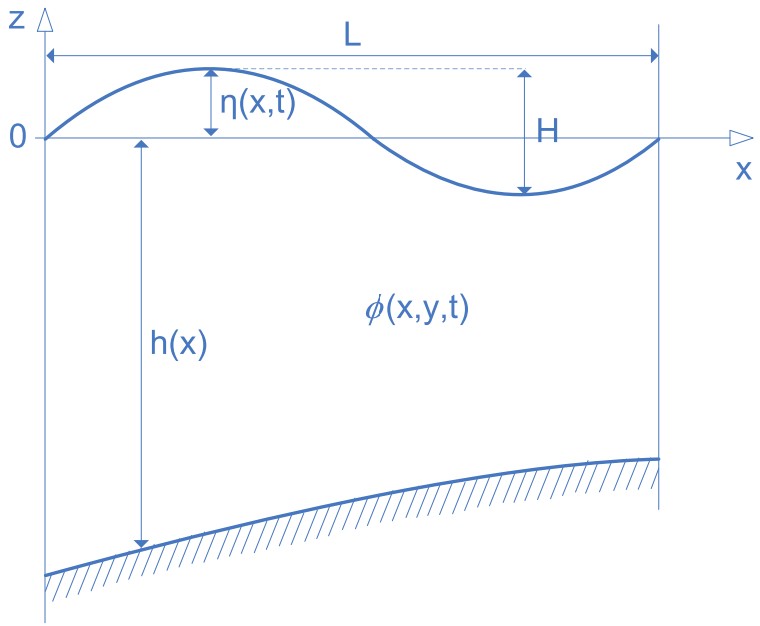

Wave parameters

- $\eta$ - surface elevation
- $\phi$ - potential ($u = \nabla\phi$)
- $h$ - still water depth
- $k = 2\pi/L$ - wave number
- $kh$ - dispersion
- $H/L$ - nonlinearity

# Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of inviscid and irrotational flow.

## 2D Potential Flow Equations




$$\partial_t \eta = -\partial_x \eta \, \partial_x \tilde{\phi} + \tilde{\omega}(1 + (\partial_x \eta)^2)$$

$$\partial_t \tilde{\phi} = -g\eta - \frac{1}{2}((\partial_x \tilde{\phi})^2 - \tilde{\omega}^2(1 + (\partial_x \eta)^2))$$

$$\tilde{\omega} = \partial_z \tilde{\phi}, \quad \tilde{\phi} = \phi|_{z=\eta}$$

For $\tilde{\omega}$ to be computed, we need to know the potential in the entire domain.

$$\phi = \tilde{\phi}, \quad z = \eta$$

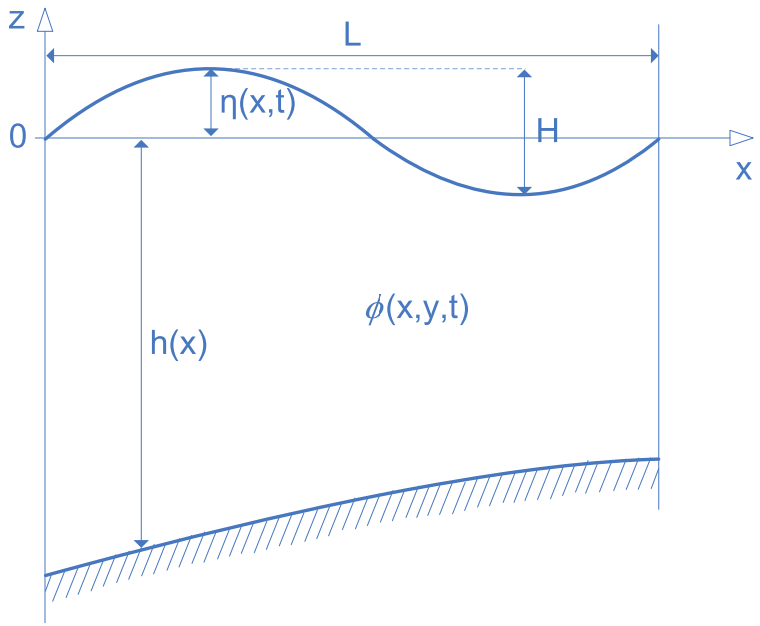
$$\partial_{xx}\phi + \partial_{zz}\phi = 0, \quad -h \leq z < \eta$$

$$\partial_z \phi + \partial_x h \, \partial_x \phi = 0, \quad z = -h$$

# Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of inviscid and irrotational flow.

## 2D Potential Flow Equations




We found that an iterative mixed-precision Defect Correction method with multigrid preconditioning solves the Laplace problem very effectively [Engsig-Karup2011] and [Glimberg2011].

# Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled as a puzzle.

```
1 // Basics
2 typedef gpulab::device_memory                                        memory_space;
3 typedef float                                                        value_type;
4 typedef gpulab::grid<value_type,memory_space>                        vector_type;
5 typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type> matrix_type;
```

# Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled as a puzzle.

```
1  // Basics
2  typedef gpulab::device_memory                                          memory_space;
3  typedef float                                                          value_type;
4  typedef gpulab::grid<value_type,memory_space>                          vector_type;
5  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type>    matrix_type;
6
7  // Multigrid setup
8  typedef gpulab::solvers::multigrid_types<
9       vector_type
10     ,matrix_type
11     ,gpulab::free_surface::jacobi_low_order_3d
12     ,gpulab::solvers::grid_handler_3d_boundary>                        multigrid_types;
13 typedef gpulab::solvers::multigrid<multigrid_types>                    preconditioner_type;
```

# Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled as a puzzle.

```
1  // Basics
2  typedef gpulab::device_memory                                          memory_space;
3  typedef float                                                          value_type;
4  typedef gpulab::grid<value_type,memory_space>                          vector_type;
5  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type>    matrix_type;
6
7  // Multigrid setup
8  typedef gpulab::solvers::multigrid_types<
9       vector_type
10     ,matrix_type
11     ,gpulab::free_surface::jacobi_low_order_3d
12     ,gpulab::solvers::grid_handler_3d_boundary>                        multigrid_types;
13 typedef gpulab::solvers::multigrid<multigrid_types>                    preconditioner_type;
14
15 // DC setup
16 typedef gpulab::solvers::defect_correction_types<
17      vector_type
18     ,matrix_type
19     ,preconditioner_type>                                              dc_types;
20 typedef gpulab::solvers::defect_correction<dc_types>                   laplace_solver_type;
```

GPUlab DTU Informatics

# Assembling the Laplace Solver

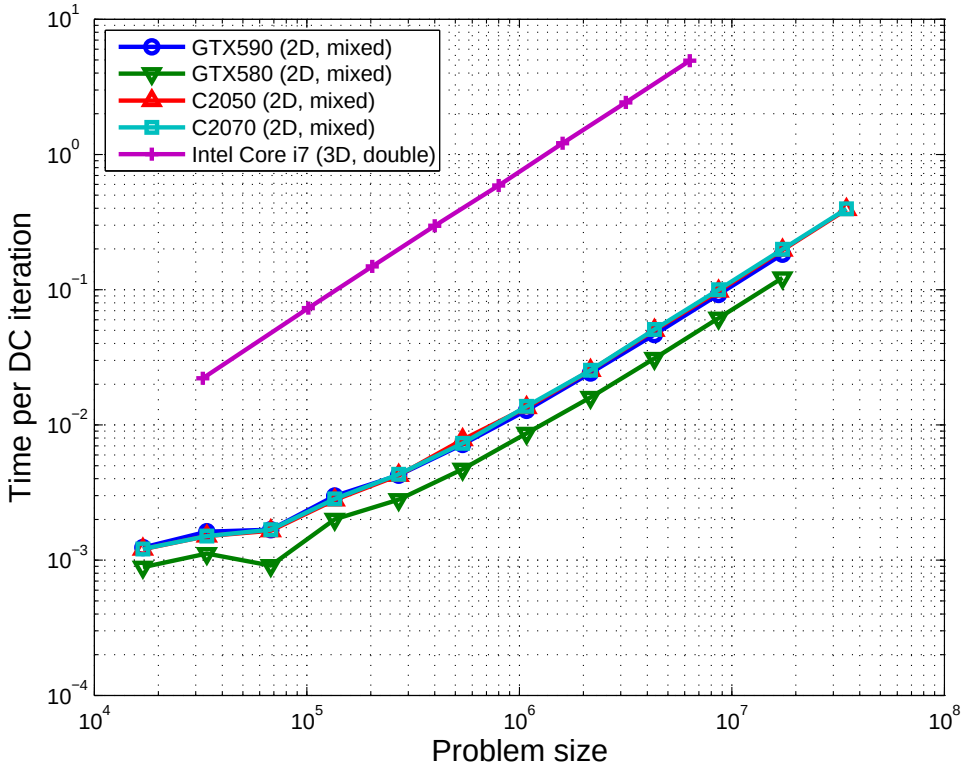
The generic nature of the library allows solvers to be assembled as a puzzle.

```
1  // Basics
2  typedef gpulab::device_memory                                          memory_space;
3  typedef float                                                          value_type;
4  typedef gpulab::grid<value_type,memory_space>                          vector_type;
5  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type>    matrix_type;
6
7  // Multigrid setup
8  typedef gpulab::solvers::multigrid_types<
9       vector_type
10     ,matrix_type
11     ,gpulab::free_surface::jacobi_low_order_3d
12     ,gpulab::solvers::grid_handler_3d_boundary>                        multigrid_types;
13 typedef gpulab::solvers::multigrid<multigrid_types>                    preconditioner_type;
14
15 // DC setup
16 typedef gpulab::solvers::defect_correction_types<
17      vector_type
18     ,matrix_type
19     ,preconditioner_type>                                              dc_types;
20 typedef gpulab::solvers::defect_correction<dc_types>                   laplace_solver_type;
```

```
1  // Somewhere in main
2  laplace_solver_type solver(A);     // Create solver
3  solver.solve(x,b);                 // Solve Ax=b
```

# 2D Performance Results

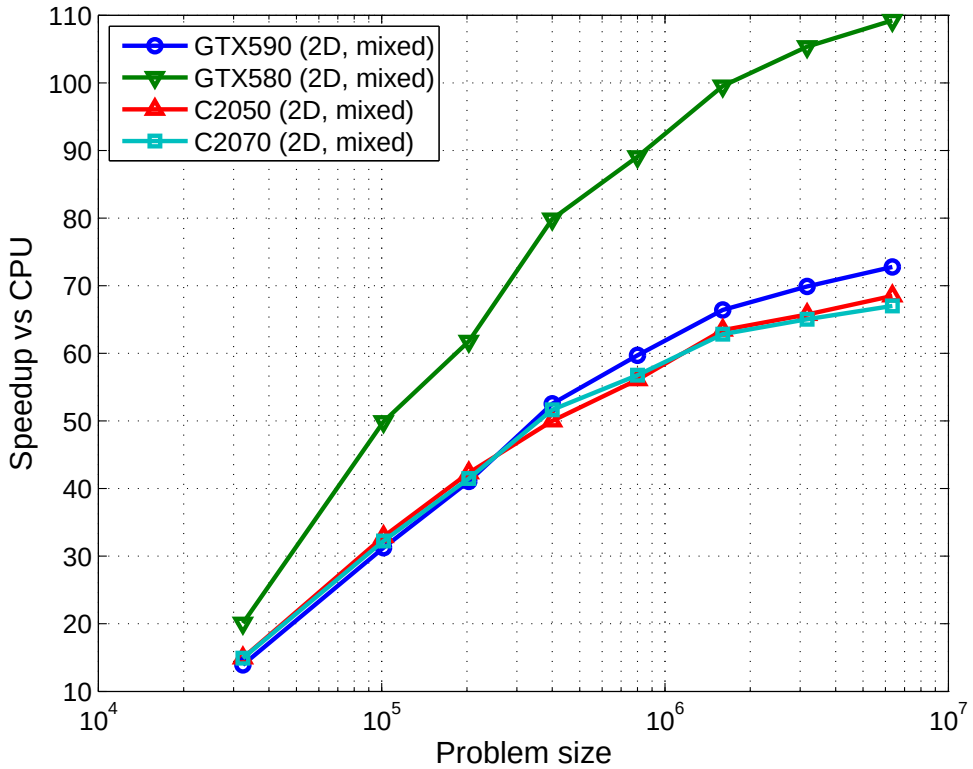
Proof-of-concept has first been established in 2D.




Timings per Defect Correction iteration. Using $6^{th}$ order accurate stencil, preconditioned with a linear $2^{nd}$ order accurate multigrid, DC+MG-RB-GS-1V(2,2).

# 2D Performance Results

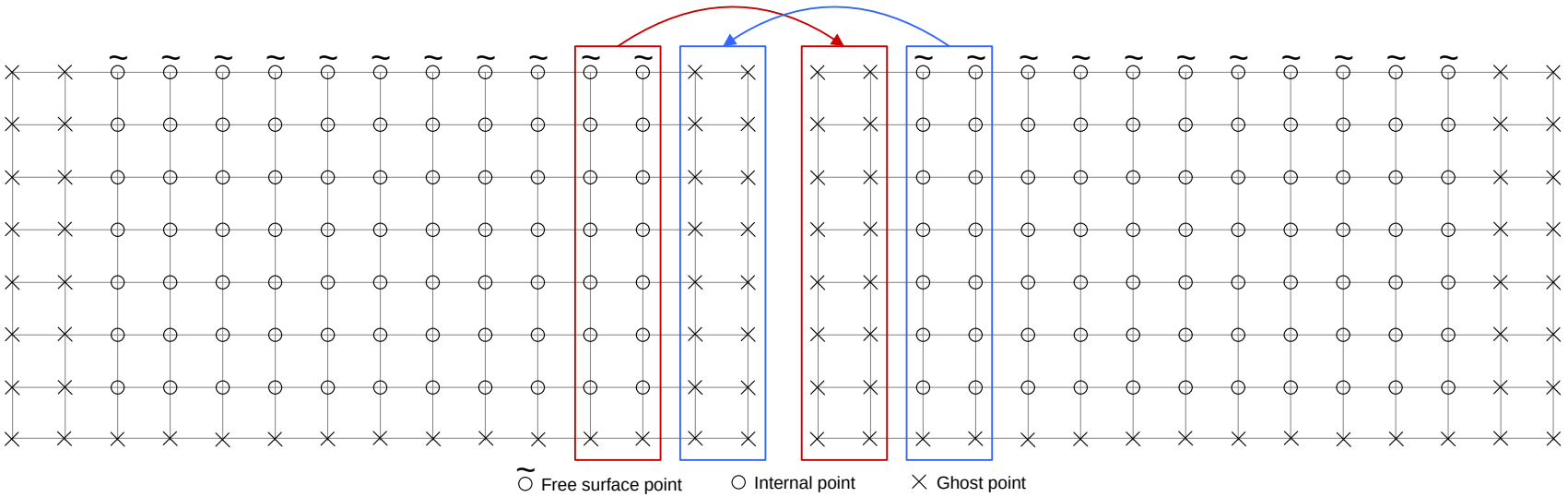
Proof-of-concept has first been established in 2D.




Timings per Defect Correction iteration. Using $6^{th}$ order accurate stencil, preconditioned with a linear $2^{nd}$ order accurate multigrid, DC+MG-RB-GS-1V(2,2).

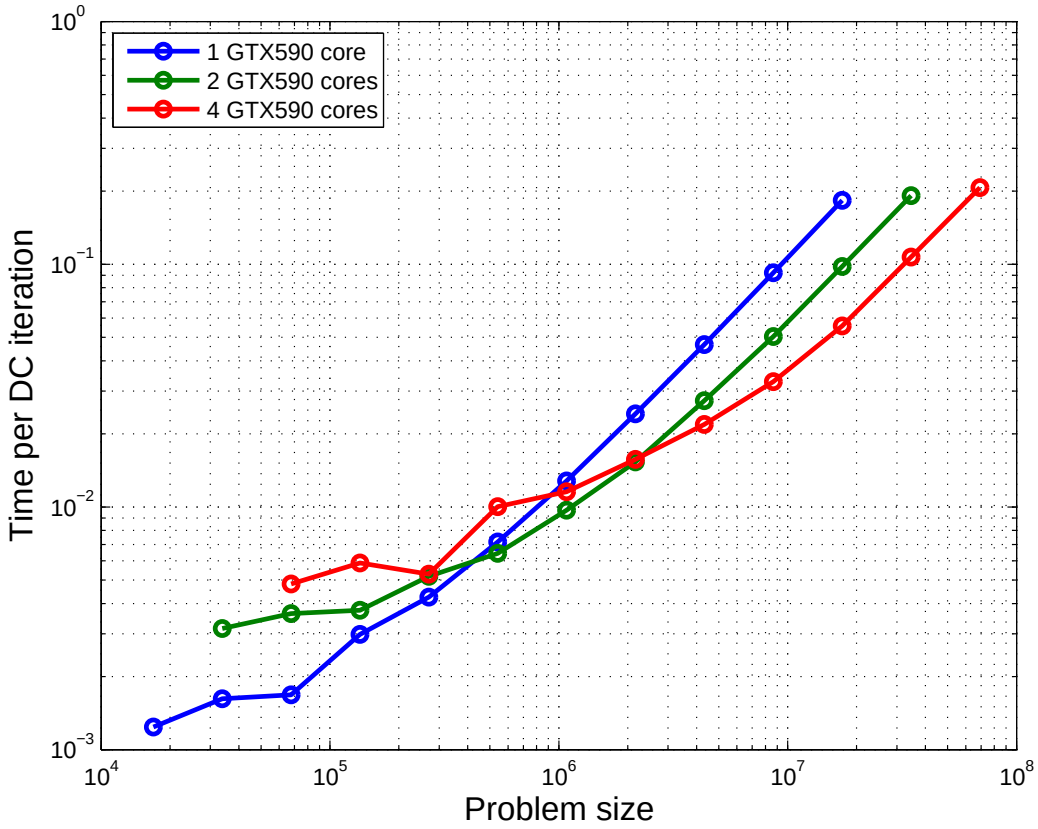
# 2D Domain Decomposition

In order to increase performance and/or resolution, we split the domain across multiple GPUs and use MPI for communication between nodes.



# 2D Domain Decomposition

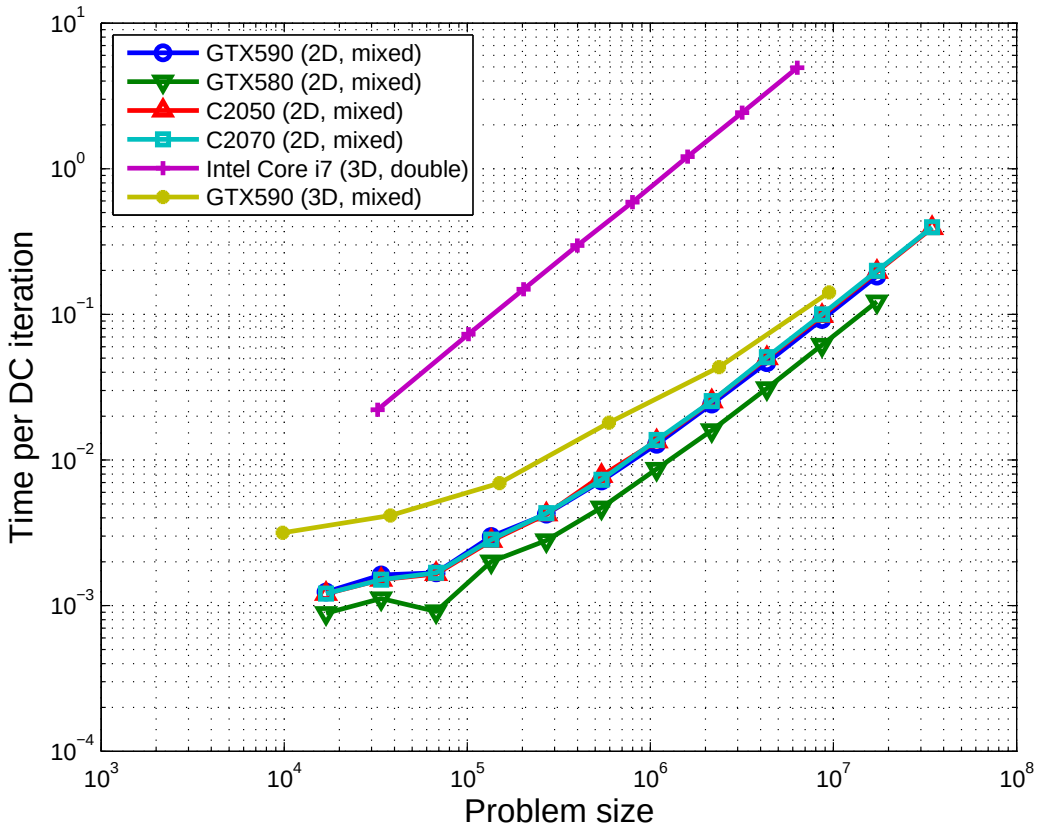
In order to increase performance and/or resolution, we split the domain across multiple GPUs and use MPI for communication between nodes.




GPUlab DTU Informatics

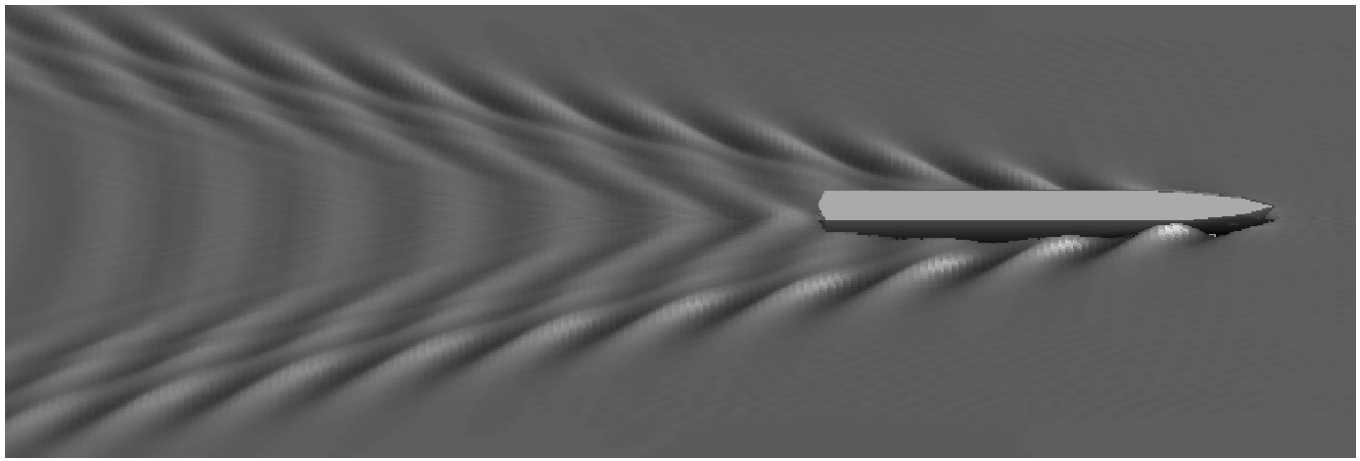
# 3D Performance Results

We are now working on a 3D version of the same solver. Recent results are not optimized, yet they show promising behavior.



# Future Work

- Add 3D domain decomposition and demonstrate scalability behavior (strong and weak scaling) at very large scales.
- Interactive tool for simulation of ship-ship interactions, used for training of naval officers.



GPUlab DTU Informatics

# Bibliography

Allan P. Engsig-Karup.

Efficient low-storage solution of unsteady fully nonlinear water waves using a defect correction method.

*Submitted to: Journal of Scientific Computing*, 2011.

Allan Peter Engsig-Karup, Morten Gorm Madsen, and Stefan Lemvig Glimberg.

A massively parallel gpu-accelerated model for analysis of fully nonlinear free surface waves.

*International Journal for Numerical Methods in Fluids*, 2011.

Stefan Lemvig Glimberg, Allan Peter Engsig-Karup, and Morten Gorm Madsen.

A fast gpu-accelerated mixed-precision strategy for fully nonlinear water wave computations.

*Proceedings of European Numerical Mathematics and Advanced Applications (ENUMATH)*, 2011.

Figures from http://www.gotw.ca/publications/concurrency-ddj.htm and www.nvidia.com